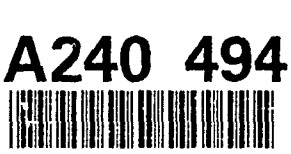

RJ 8180 (75139) June 19, 1991
Computer Science

Word Problems – This Time with Interleaving

Alain J. Mayer[1]
Dept. of Computer Science
Brown University
Providence, RI 02912.

Larry J. Stockmeyer
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Abstract. We consider regular expressions extended with the interleaving operator, and investigate the complexity of membership and inequivalence problems for these expressions. For expressions using the operators union, concatenation, Kleene star, and interleaving, we show that the inequivalence problem (deciding whether two given expressions do not describe the same set of words) is complete for exponential space. Without Kleene star, we show that the inequivalence problem is complete for the class Σ_2^p at the second level of the polynomial-time hierarchy. Certain cases of the membership problem (deciding whether a given word is in the language described by a given expression) are shown to be NP-complete. Special cases of the membership problem which can be solved in polynomial time are also discussed.

[1]The research of this author was partly supported by ONR grant N00014-91-J-1613. Part of this work was performed while the author was at the IBM T.J. Watson Research Center.

1 Introduction

There has been considerable progress in classifying the the computational complexity of decision problems involving "regular-like" expressions. Such expressions are similar to the Kleene regular expressions of finite-automata theory, but may contain operators on sets of words other than the usual operators union, concatenation, and star. Problems which have been studied include *inequivalence*, i.e., deciding whether two given expressions do not describe the same set of words, and *membership*, i.e., deciding whether a given word is in the language described by a given expression. Previous work on this subject can be found, for example, in [Furer80, Hunt73, HRS76, Stock74, StM73]; see also [AHU74, HU79]. In particular, we focus here on the *interleaving* operator. The interleaving of words x and y, denoted $x|y$, is the set of all words of the form

$$x_1 y_1 x_2 y_2 \ldots x_k y_k$$

where $k > 0$, $x = x_1 x_2 \ldots x_k$, $y = y_1 y_2 \ldots y_k$, and where the words x_i and y_i, $1 \leq i \leq k$, can be of arbitrary length (including the empty word).

The motivation to investigate the interleaving operator is twofold. First, the interleaving operator can be interpreted as the simplest case of the *composition* operator used in algebraic approaches to modeling concurrent computation. Interleaving represents the case where processes run concurrently in such a fashion that their atomic steps can be arbitrarly interleaved but where no communication between them takes place. One of the best known formalisms for specifying and verifying concurrent systems is CCS (see [Miln80]). In [Miln84] a restricted set of algebraic operators (i.e. $\{\cdot, \cup, *\}$) is used to form the *star expressions* in CCS. These expressions are syntactically identical to regular expressions, but instead of having as semantics "sets of strings", their semantics is "equivalence classes of processes". In [KS90] it is shown that the observational equivalence problem of star expressions is solvable in polynomial time. We believe that the techniques presented in this paper will be useful to determine the complexity of the observational equivalence problem of star expressions extended by a suitably defined composition operator. This is an open question of [KS90].

Secondly, as we discovered while doing this work, the interleaving operator has some interesting properties of its own: **Succinctness:** The use of the interleaving operator can shorten a regular expression by an exponential amount. **Simulation of Integer Addition and Intersection:** Under certain format restrictions, addition of positive integers and intersection of expressions can be simulated by the use of the interleaving operator. **Complexity:** The inequivalence problem for expressions with interleaving, but without star, is one of the few natural problems known to be Σ_2^p-complete.

We now outline the remainder of the paper. Definition are given in Section 2. In Section 3, we present a language for which a succinct expression with interleaving exists but every regular expression is longer by an exponential factor. Section 4 illustrates the nature of the interleaving operator via the membership problem restricted to expressions containing a constant number of interleavings. In Section 5, we show certain cases of the membership problem to be NP-complete. One such case is the problem of determining, given words $z, u_1, \ldots, u_n$ (with n variable), whether z can be written as an interleaving of $u_1, \ldots, u_n$. Sections 6 and 7 are devoted to the inequivalence problem for expressions without and

with the Kleene star, respectively. In the case without star, we show that interleaving is powerful enough to simulate addition of integers under certain format restrictions. We can then emulate a proof of [Stock77] to show that the inequivalence problem is Σ_2^p-complete. In the case with star, we show that interleaving can simulate intersection, again under appropriate format restrictions. We can then emulate a proof of [Furer80] to show that the inequivalence problem is exponential-space-complete.

2 Definitions

Basic familiarity with regular expressions, time and space complexity, polynomial-time reducibility, and complete problems is assumed. The necessary background, if needed, can be found in [AHU74] or [HU79], for example.

We now define more precisely the types of expressions and problems of interest. Let ϵ denote the empty word. Let Σ be a finite alphabet and let S be a subset of the operators $\{\cup, \cdot, *, \cap, |\}$. We define the *S-expressions* (over Σ) and simultaneously define the operator L which maps each S-expression to a subset of Σ^*:

1. For every $\sigma \in \Sigma \cup \{\epsilon\}$, σ is an S-expression, and $L(\sigma) = \{\sigma\}$;
2. If r_1 and r_2 are S-expressions and $@ \in S - \{*\}$, then $(r_1 @ r_2)$ is an S-expression, and $L((r_1 @ r_2)) = L(r_1) @ L(r_2)$;
3. If r is an S-expression, then (r^*) is an S-expression, and $L((r^*)) = (L(r))^*$.

In 2, the interleaving operator is extended to sets of words in the obvious way, i.e., $L_1|L_2$ is the union of the sets $w_1|w_2$ taken over all $w_1 \in L_1$ and $w_2 \in L_2$. When writing expressions in the text, extraneous parentheses are often omitted. Although it is sometimes convenient to use ϵ when writing expressions, our results do not change if expressions cannot contain ϵ.

Letting S be as above, the problem MEMBER-S is the problem of deciding, given an S-expression r and a word $w \in \Sigma^*$, whether $w \in L(r)$. The problem INEQ-S is the problem of deciding, given two S-expressions r_1 and r_2, whether $L(r_1) \neq L(r_2)$. The problem NEC-S is a special case of INEQ-S; here the problem is to decide, for a given r, whether $L(r) \neq \Sigma^*$.

$|w|$ denotes the length of the word w, and $|r|$ denotes the length of the expression r.

It will be useful to define $|$ also as an operator on nondeterministic finite automata (NFA's) M_1 and M_2 in such a way that $L(M_1 \mid M_2) = L(M_1) \mid L(M_2)$. Here is the relevant definition (see Eilen74]):

> Let $M_i = (Q_i, \Sigma, \delta_i, p_{0i}, F_i)$ $(i = 1, 2)$ be an NFA with (in notation of [AHU74]) state set Q_i, input alphabet Σ, transition function δ_i, initial state p_{0i}, and accepting states F_i. Then $M = M_1 \mid M_2$ is defined as follows:
> $M = (Q_1 \times Q_2, \Sigma, \delta, [p_{01}, p_{02}], F)$ where the new transition relation δ is defined as $\delta([q_1, q_2], a) = (\delta_1(q_1, a) \times \{q_2\}) \cup (\{q_1\} \times \delta_2(q_2, a))$, and the new set of accepting states F is defined by $F([q_1, q_2]) = F_1(q_1) \wedge F_2(q_2)$.

Note that the number of states of $M_1 \mid M_2$ is the product of the number of states of M_1 and the number of states of M_2.

3 On the Expressiveness of Interleaving

In this section, we will show an example in which the use of the interleaving operator shortens a regular expression by an exponential amount. Consider the alphabet $\Sigma_n = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ and the language L_n of all permutations of length n in Σ_n^*, i.e., L_n is the set of words of length n in which each symbol σ_i appears exactly once. Obviously, we have $L_n = L(\sigma_1|\sigma_2|\ldots|\sigma_n)$. But as the following proposition shows, there is no standard regular expression of polynomial length denoting L_n.

Proposition 3.1 *Every $\{\cup, \cdot, ^*\}$-expression r with $L(r) = L_n$ has $|r| = \Omega(2^n)$.*

PROOF: For $S \subseteq \Sigma_n$, let the word $w(S)$ be the concatenation of the symbols in S in order of increasing index. Let $\overline{S}$ denote the complement of S with respect to Σ_n. Note that for any S, $w(S)w(\overline{S}) \in L_n$. We claim that the number of subsets of Σ_n, namely 2^n, is a lower bound on the number of states of any NFA accepting L_n.

Assume that there is an NFA M with fewer states. Then there are two subsets S and T with $S \neq T$ and a state q such that there is a computation path of M on input $w(S)$ from the start state to q, a path on input $w(T)$ from the start state to q, a path on input $w(\overline{S})$ from q to an accepting state, and a path on input $w(\overline{T})$ from q to an accepting state. Assuming (without loss of generality) that S is not a subset of T, there must be a σ_j in S which is not in T, so σ_j is in $\overline{T}$. Therefore, M accepts the word $w(S)w(\overline{T})$ which contains two occurrences of σ_j. Thus any NFA accepting L_n must have at least 2^n states. Since any regular expression r can be converted to an equivalent NFA having $O(|r|)$ states, the proposition follows. □

4 Example: Interleaving of a Constant Number of Strings

To illustrate the interleaving-operator we show how to answer the following question in a straightforward way: $z \in L(u_1 \mid u_2 \mid \ldots \mid u_k)$?, where k is a constant, ($1 \leq i \leq k$) and z are strings over some alphabet Σ, and $|u_i| = n$ and $|z| = kn$.

1. Construct the NFA M for $u_1 \mid u_2 \mid \ldots \mid u_k$. Its transition diagram will be an $n \times n \ldots \times n$ (k times) grid of states. Thus we can think of it as a k-dimensional hypercube of side n. Assume that the start state is at the "upper left" corner and the only accepting state (s_a) is at the "lower right" corner. Every state has at most k successors, each of which has one coordinate closer to s_a. Thus there are $O(n^k)$ states and $O(n^k)$ transitions. Note that every path from the starting state to the accepting state has length kn.

2. Let S be a set of states. Simulate M on input z by storing in S the states which M can reach after reading the prefix of z consumed so far. After reading at most kn symbols one of the following two conditions will become true: (i) $S = \{\}$ and thus REJECT or (ii) $S = \{s_a\}$ and thus ACCEPT. Note that the size of S can be at most $O(n^{k-1})$, since there are at most $O(n^{k-1})$ states at distance l ($1 \leq l \leq kn$) from the start state.

It can be easily verified that the above procedure can be carried out using $O(n^k)$ .ime and $O(n^{k-1})$ space on a unit-cost RAM. In [vLN82] a dynamic programming algorithm was used to improve the time performance to $O(n^k/\log^{1/(k-1)} n)$. This was further improved by [IPC85] to $O(n^k/\log^{k/(k-1)} n)$.

This result is easily generalized to the following.

Theorem 4.1 *For each constant k, the problem MEMBER-$\{\cup, \cdot, *, |\}$, restricted to expressions having at most k occurrences of $|$, can be solved in polynomial time.*

PROOF: Given a word z and a $\{\cup, \cdot, *, |\}$-expression r, there is an NFA having $O(|r|^k)$ states which accepts $L(r)$. The NFA is constructed as in [HU79, Chap. 2], where interleavings are handled by the construction described in Section 2. The NFA is then simulated on input z as described above. □

5 Instances of MEMBER which are $\mathcal{NP}$-Complete

The problem MEMBER-$\{\cup, \cdot, *, \cap\}$ is known to be solvable in polynomial time (see the solution to Problem 3.23 in [HU79]). We show in this section that the problem MEMBER-$\{\cup, \cdot, *, \cap, |\}$ is $\mathcal{NP}$-complete. In fact, we will prove an even stronger result by showing that the following problem SHUFFLE is $\mathcal{NP}$-hard:

> An instance of SHUFFLE consists of $n+1$ words $z, u_1, \ldots, u_n$ for some n, and the question is whether $z \in L(u_1 \mid \ldots \mid u_n)$.

(This is the problem of the last section where the number of strings (k) can be variable.) We also show that MEMBER-$\{\cup, \cdot, \cap, |\}$ is $\mathcal{NP}$-hard even if $|$ is used only once in the expression.

Theorem 5.1 *MEMBER-$\{\cup, \cdot, *, \cap, |\}$ is $\mathcal{NP}$-complete. The problem remains $\mathcal{NP}$-hard even if only $\{\cdot, |\}$ are used in the expression, or if $\{\cup, \cdot, \cap, |\}$ are used in the expression and $|$ appears only once. Also, these problems remain $\mathcal{NP}$-hard if an alphabet Σ of size 3 is used.*

We will now prove this theorem by a sequence of lemmas.

Lemma 5.2 *SHUFFLE is $\mathcal{NP}$-hard.*

PROOF: We will prove this lemma by doing a reduction from the well known $\mathcal{NP}$-hard 3-dimensional matching problem:

> Given disjoint sets $W = \{w_1, w_2, \ldots, w_q\}$, $X = \{x_1, x_2, \ldots, x_q\}$, $Y = \{y_1, y_2, \ldots, y_q\}$, and given a set $M \subseteq W \times X \times Y$, say $M = \{m_1, m_2, \ldots, m_s\}$, does M have a matching? I.e., does there exist a set $M' \subseteq M$ in which every element of $W \cup X \cup Y$ appears exactly once?

Let c be a symbol not in $W \cup X \cup Y$. We will construct strings $z, \mu_1, \ldots, \mu_k$ over the alphabet $\Sigma = W \cup X \cup Y \cup \{c\}$, such that

$$z \in L(\mu_1 \mid \mu_2 \mid \ldots \mid \mu_k) \text{ iff } M \text{ contains a matching } M'.$$

We will use the following notation: $f_j(i)$ $(1 \leq j \leq 3,\ 1 \leq i \leq k)$ denotes the index of the jth component of m_i. Define $n(w_j)$ $(n(x_j), n(y_j))$ to be the total number of occurrences of the element w_j (x_j, y_j) in all of the elements of M.

Now define:

$$\mu_i = w_{f_1(i)}\, x_{f_2(i)}\, y_{f_3(i)}\, c$$

$$r = \mu_1 \mid \mu_2 \mid \ldots \mid \mu_k$$

$$z = w_1\, w_2 \ldots w_q\, x_1\, x_2 \ldots x_q\, y_1\, y_2 \ldots y_q\, c^q\, w_1^{n(w_1)-1}\, w_2^{n(w_2)-1} \ldots y_q^{n(y_q)-1}\, c^{k-q}.$$

(1) M contains a matching $\Rightarrow z \in L(r)$:
Let M' be a matching. Let $g(i)$ $(1 \leq i \leq q)$ denote the ith element in M'. Since M' is a matching, there is a way to interleave $\mu_{g(1)}, \ldots, \mu_{g(q)}$ to obtain the first $4q$ symbols of z. The rest of z can be trivially obtained.

(2) $z \in L(r) \Rightarrow M$ contains a matching:
The only way to choose the interleaving to obtain the first $4q$ symbols of z is to interleave q whole μ's. The corresponding elements of M thus form a matching. □

Lemma 5.3 *MEMBER-$\{\cup, \cdot, \cap, |\}$ is $\mathcal{NP}$-hard even if $|$ appears just once in the expression.*

PROOF: We will prove this lemma by doing a reduction from the well known $\mathcal{NP}$-hard problem 3SAT. Assume that we are given a formula $C = \{c_1, c_2, \ldots, c_m\}$ as a collection of m clauses on a finite set $\{v_1, v_2, \ldots, v_n\}$ of variables such that $|c_i| = 3$ $(1 \leq i \leq m)$.

We will use the following notation: p_i $(1 \leq i \leq m)$ is the set of indices of the variables appearing positively in c_i, and n_i $(1 \leq i \leq m)$ is the set of indices of the variables appearing negatively in c_i.

We construct a string z and an expression r over the alphabet $\Sigma = \{v_1, v_2, \ldots, v_n\}$ such that

$$z \in L(r) \text{ iff } C \text{ is satisfiable.}$$

Let C_i $(1 \leq i \leq m)$ be the regular expression defined as follows:

$$C_i = \bigcup_{k \in n_i} ((\Sigma - v_k) \cup \epsilon)^n \ \cup\ ((\Sigma \cup \epsilon)^n \cdot (\bigcup_{l \in p_i} v_l) \cdot (\Sigma \cup \epsilon)^n).$$

Thus $C_i \cap (\Sigma \cup \epsilon)^n$ contains exactly all words of length at most n in which (1) at least one symbol whose index is in n_i does not appear, or in which (2) at least one symbol whose index is in p_i appears. Now let r be defined as:

$$r = (C_1 \cap C_2 \cap \ldots \cap C_m) \mid (\Sigma \cup \epsilon)^n$$

and z as:

$$z = v_1\, v_2 \ldots v_n.$$

(1) C is satisfiable $\Rightarrow z \in L(r)$:
Let T be a satisfying truth assignment for C. Let the partitioning of z be such that the

symbol v_j belongs to the LIS of "|" iff $T(v_j) = 1$. In other words $z = x_1y_1 \ldots x_ky_k$, where $x = x_1x_2 \ldots x_k$ is exactly the sequence of all variables true under T in ascending order. Since T is satisfying, we know that for all i $(1 \leq i \leq m)$ the word x (with $|x| \leq n$) either (i) contains at least one symbol with index in p_i or (ii) does not contain all the symbols with index in n_i. From this it easily follows that x is an element of every C_i $(1 \leq i \leq m)$ and we are done.

(2) $z \in L(\tau) \Rightarrow C$ is satisfiable:
Let the partitioning of z, by which its membership in $L(\tau)$ is shown, be $z = x_1y_1 \ldots x_ky_k$. Thus the word $x = x_1x_2 \ldots x_k$ is a member of every C_i $(1 \leq i \leq m)$. Thus we can define a truth assignment:

$$T(v_j) = \begin{cases} 1 & \text{if } v_j \in x \\ 0 & \text{if } v_j \notin x\ . \end{cases}$$

T obviously satisfies every clause. □

Lemma 5.4 *In Lemmas 5.2 and 5.3 we can use an alphabet of size 3 instead of an alphabet of variable size.*

PROOF: We code all symbols involved in the following way. Let h be a one-to-one mapping from Σ to the positive integers. Then we can code every σ in Σ as "$\#\, 1^{h(\sigma)}\, \&$". Correct interleavings are now interleavings in which blocks representing one symbol are never separated. It is easy to see that correct interleavings are uniquely readable and incorrect interleavings can be easily detected. □

We now prove the $\mathcal{NP}$ upper bound.

Lemma 5.5 *MEMBER-*$\{\cup, \cdot, *, \cap, |\}$ *is in* $\mathcal{NP}$.

PROOF: Let z be a word in Σ^* and let E be an expression over Σ. We define a "proof" that $z \in L(E)$ recursively as follows. First, if $z = \epsilon$, then the symbol e is a proof of (z, E) if $\epsilon \in L(E)$. In the remaining cases, we assume $z \neq \epsilon$. (i) If $z \in \Sigma$, then z is a proof of (z, z); (ii) if P_1 is a proof of (z_1, E_1), P_2 is a proof of (z_2, E_2), and $z = z_1 \cdot z_2$, then $(z, P_1 \cdot P_2)$ is a proof of $(z, (E_1 \cdot E_2))$; (iii) if P is a proof of (z, E) then P is a proof of $(z, (E \cup E'))$ and of $(z, (E' \cup E))$ for any expression E'; (iv) if P_1 is a proof of (z, E_1) and P_2 is a proof of (z, E_2), then $(z, P_1 \cap P_2)$ is a proof of $(z, (E_1 \cap E_2))$; (v) if P_1 is a proof of (z_1, E_1), P_2 is a proof of (z_2, E_2), and $z \in L(z_1 \mid z_2)$, then $(z, P_1 \mid P_2)$ is a proof of $(z, (E_1 \mid E_2))$; (vi) if $z = z_1z_2 \ldots z_k$ for some $k \geq 1$ and words $z_i \neq \epsilon$ for $1 \leq i \leq k$, and if P_i is a proof of (z_i, E) for $1 \leq i \leq k$, then $(z, P_1, \ldots, P_k)$ is a proof of $(z, (E^*))$.

Let Q be the relation $Q(z, E, P)$ iff P is a proof of (z, E). The question "$\epsilon \in L(E)$?" can be solved in polynomial time. Since also the question "$z \in L(z_1 \mid z_2)$?" can be solved in polynomial time (see Section 4), it is easy to see that Q can be computed in polynomial time. By induction on the structure of E it is not hard to verify that, if P is a proof of (z, E) and $z \neq \epsilon$, then $|P| \leq 2|z||E|$. We illustrate the induction step for case (vi) (star):

$$
\begin{aligned}
|P| &\leq |z| + k + 2 + \sum_{i=1}^{k} |P_i| \\
&\leq |z| + k + 2 + \sum_{i=1}^{k} 2|z_i||E| \qquad \text{by induction} \\
&\leq 2|z| + 2 + 2|z||E| \qquad \text{since } k \leq |z| \text{ and } z = z_1 \ldots z_k \\
&\leq 2|z|(|E| + 3) \qquad \text{since } |z| \geq 1 \\
&= 2|z||(E^*)|.
\end{aligned}
$$

Now we can write $z \in L(E)$ iff $(\exists P : Q(z, E, P))$. It follows that the membership problem belongs to $\mathcal{NP}$. □

Lemmas 5.2 – 5.5 prove Theorem 5.1

6. Inequivalence for Expressions without Star

There are few natural problems known to be complete in the class Σ_2^p of the polynomial-time hierarchy [Stock77]. In this section, we add another problem to this list by showing that INEQ-$\{\cup, \cdot, |\}$ is Σ_2^p-complete. The proof will make use of the fact that interleaving is powerful enough to simulate addition of positive integers.

Theorem 6.1 *INEQ-$\{\cdot, \cup, |\}$ is Σ_2^p-complete.*

PROOF: We will prove this theorem in two parts. First that the problem belongs to Σ_2^p, and then that it is Σ_2^p-hard. Both parts of this proof are similar to the proof that the inequivalence problem for integer expressions is Σ_2^p-complete [Stock77].

Membership
By induction on the structure of E, it is easy to show that, if E is a $\{\cup, \cdot, |\}$-expression and $z \in L(E)$, then $|z| \leq |E|$. Let the notion of a "proof" and the predicate Q be defined as in the proof of Lemma 5.5. Then we can write:

$$(E_1, E_2) \in \text{INEQ}\{\cup, \cdot, |\} \text{ iff } (\exists z : (\exists P_1 : Q(z, E_1, P_1)) \leftrightarrow \neg(\exists P_2 : Q(z, E_2, P_2))).$$

Standard manipulation of quantifiers and Theorem 3.1 of [Stock77] imply now that INEQ-$\{\cup, \cdot, |\}$ is in Σ_2^p.

Hardness
We first show that, using certain format requirements, we can simulate addition of positive integers by interleaving.

Let $a_{i,k}$, for $1 \leq i \leq n+1$ and $1 \leq k \leq m$, be positive integers. For each k, let s_k be the sum of $a_{i,k}$ for $1 \leq i \leq n+1$. Let E be the expression:

$$
\begin{aligned}
E = \; & 1^{a_{1,1}} \cdot \dot{\flat} \cdot 1^{a_{1,2}} \cdot \dot{\flat} \cdots 1^{a_{1,m}} \cdot \dot{\flat} \; | \\
& 1^{a_{2,1}} \cdot \dot{\flat} \cdot 1^{a_{2,2}} \cdot \dot{\flat} \cdots 1^{a_{2,m}} \cdot \dot{\flat} \; | \\
& \cdots \\
& 1^{a_{n+1,1}} \cdot \dot{\flat} \cdot 1^{a_{n+1,2}} \cdot \dot{\flat} \cdots 1^{a_{n+1,m}} \cdot \dot{\flat}.
\end{aligned}
$$

Let R be the set of words over alphabet $\{1, b\}$ such that every block of consecutive b's has length at least $n+1$.

Lemma 6.2 *$L(E) \cap R$ contains the single word* $1^{s_1} \cdot b^{n+1} \cdot 1^{s_2} \cdot b^{n+1} \cdots 1^{s_m} \cdot b^{n+1}$.

PROOF: For a word in $L(E)$ the only way to build $n+1$ consecutive b's is to first interleave all leading 1's from all $n+1$ arguments of the interleaving operator (i.e., $1^{a_{1,1}}, 1^{a_{2,1}}, \ldots, 1^{a_{n+1,1}}$), and then all first b's of the arguments, etc. □

We now show the desired hardness result by doing a reduction from $(B_2 \cap \text{DNF})$, which is shown to be Σ_2^p-hard in [Stock77, Wrath77]. An instance of $(B_2 \cap \text{DNF})$ is a Boolean formula $G(X_1, X_2)$ where X_j $(j = 1, 2)$ is a set of variables $\{x_{j1}, x_{j2}, \ldots, x_{jn}\}$, and where G is in disjunctive normal form, i.e., $G = C_1 \vee C_2 \vee \ldots \vee C_m$, where each C_k is a conjunction of literals; the question is whether $\exists X_1 \forall X_2 (G(X_1, X_2) = 1)$. We can assume that a variable and its negation do not both appear in the same clause.

In order to show the Σ_2^p-hardness of INEQ-$\{\cdot, \cup, |\}$ we will construct expressions E_1, E_2, and $\overline{R}$ such that

$$\begin{gathered} \forall X_1 \exists X_2 (G(X_1, X_2) = 0) \\ \text{iff} \\ L(E_1 \cup \overline{R}) \subseteq L(E_2 \cup \overline{R}). \end{gathered} \tag{1}$$

Letting R be defined as in Lemma 6.2, the expression $\overline{R}$ will have the properties that $L(\overline{R}) \cap R = \emptyset$ and $L(E_1) - R \subseteq L(\overline{R})$. It is easy to verify that these two properties imply that

$$L(E_1 \cup \overline{R}) \subseteq L(E_2 \cup \overline{R}) \text{ iff } L(E_1) \cap R \subseteq L(E_2) \cap R.$$

Therefore, to prove (1) it suffices to show

$$\begin{gathered} \forall X_1 \exists X_2 (G(X_1, X_2) = 0) \\ \text{iff} \\ L(E_1) \cap R \subseteq L(E_2) \cap R. \end{gathered}$$

Let us first define $\overline{R}$. Since all words in $L(E_1)$ are bounded in length by $M := |E_1|$, the following will do:

$$\overline{R} \;=\; ((b \cup 1 \cup \epsilon)^{M} \cdot 1 \cup \epsilon) \cdot b \cdot (b \cup \epsilon)^{n-1} \cdot (1 \cdot (b \cup 1 \cup \epsilon)^{M} \cup \epsilon).$$

Let $[\ldots]$ be 11 if the expression in the brackets is true and 1 if it is false. Let $\overline{[\ldots]}$ be the opposite.

The expression E_1 is now:

$$\begin{aligned} E_1 \;=\; & (\overline{[x_{11} \in C_1]} \cdot b \cdot \overline{[x_{11} \in C_2]} \cdot b \cdots \overline{[x_{11} \in C_m]} \cdot b \\ & \qquad \cup \overline{[\neg x_{11} \in C_1]} \cdot b \cdot \overline{[\neg x_{11} \in C_2]} \cdot b \cdots \overline{[\neg x_{11} \in C_m]} \cdot b) \mid \\ & \text{interleaved with similar subexpressions for } x_{12}, \ldots, x_{1n} \mid \\ & 1^{n+1} \cdot b \cdot 1^{n+1} \cdot b \cdots 1^{n+1} \cdot b. \end{aligned}$$

(The last subexpression contains m repetitions of $1^{n+1} \cdot b$.)

Let F be $(1 \cup 1^2 \cup 1^3 \cup \ldots \cup 1^{2n})$.

The expression E_2 is:

$$E_2 = ([x_{21} \in C_1] \cdot b \cdot [x_{21} \in C_2] \cdot b \cdots [x_{21} \in C_m] \cdot b \cup [\neg x_{21} \in C_1] \cdot b \cdot [\neg x_{21} \in C_2] \cdot b \cdots [\neg x_{21} \in C_m] \cdot b) \mid$$
interleaved with similar subexpressions for $x_{22}, \ldots, x_{2n}$ $\mid$
$F \cdot b \cdot F \cdot b \cdots F \cdot b.$

If we now restrict the words in $L(E_1)$ and $L(E_2)$ to be in R, we can use Lemma 6.2 to conclude that all words in $L(E_1) \cap R$ and in $L(E_2) \cap R$ are of the form

$$y = 1^{s_1} \cdot b^{n+1} \cdot 1^{s_2} \cdot b^{n+1} \cdots 1^{s_m} \cdot b^{n+1}.$$

It is useful to write numerical expressions for the numbers s_k, $1 \leq k \leq m$. For words in $L(E_1) \cap R$, the expressions are functions of 0-1 valued variables p_{1i} for $1 \leq i \leq n$. Setting $p_{1i} = 0$ (resp., $p_{1i} = 1$) means that we choose the LHS (resp., RHS) of the ith union in E_1 to produce the corresponding word in $L(E_1) \cap R$. We also interpret $[\ldots]$ as being either 1 or 2 (as opposed to 1 or 11). Now $y \in L(E_1) \cap R$ iff there are $p_{1i} \in \{0,1\}$ such that, for $1 \leq k \leq m$,

$$s_k = \sum_{i=1}^{n} ((1 - p_{1i})\overline{[x_{1i} \in C_k]} + p_{1i}\overline{[\neg x_{1i} \in C_k]}) + (n + 1).$$

The numerical expressions for words in $L(E_2) \cap R$ involve 0-1 valued variables p_{2i} which, as above, indicate whether the LHS or RHS of each union in E_2 is used. These expressions also involve variables f_k for $1 \leq k \leq m$, where $1 \leq f_k \leq 2n$ for all k; here f_k indicates which word is taken from the kth occurrence of F in E_2. Now $y \in L(E_2) \cap R$ iff there are $p_{2i} \in \{0,1\}$ and $f_k \in \{1, 2, \ldots, 2n\}$ such that, for $1 \leq k \leq m$,

$$s_k = \sum_{i=1}^{n} ((1 - p_{2i})[x_{2i} \in C_k] + p_{2i}[\neg x_{2i} \in C_k]) + f_k.$$

We now can, as in [Stock77], identify four facts about E_1 and E_2. The following terminology is used. If X is a set of variables, an *X-assignment* is an assignment of truth values to the variables in X. We say that an X-assignment α *kills* the clause C_k if either some literal z appears in C_k and z is assigned value *false* by α or some literal $\neg z$ appears in C_k and z is assigned value *true* by α.

(a) For each $y \in L(E_1) \cap R$, $2n + 1 \leq s_k \leq 3n + 1$ for $1 \leq k \leq m$; and there is an X_1-assignment such that, for $1 \leq k \leq m$, $s_k = 3n + 1$ iff the assignment does not kill C_k.

(b) For each X_1-assignment there is a $y \in L(E_1) \cap R$ such that, for $1 \leq k \leq m$, $s_k = 3n + 1$ iff the assignment does not kill C_k.

(c) For each $y \in L(E_2) \cap R$ there is an X_2-assignment such that, for $1 \leq k \leq m$, if $s_k = 3n + 1$ then the assignment kills C_k.

(d) Let A_2 be an X_2-assignment and y be a word over $\{1,b\}^*$ having the form $1^{s_1} \cdot b^{n+1} \cdot 1^{s_2} \cdot b^{n+1} \ldots$ such that $2n+1 \leq s_k \leq 3n+1$ and $(s_k = 3n+1) \Rightarrow (A_2$ kills $C_k)$ for $1 \leq k \leq m$. Then $y \in L(E_2) \cap R$.

The proofs of (a)-(d) are not difficult. In each case, we must draw a correspondence between a truth assignment and a word y. As just noted, each word corresponds to values for the 0-1 variables p_{1i} or p_{2i}. The correspondence between these variables and the Boolean variables in G is that $p_{ji} = 1$ iff x_{ji} is assigned value *true*. We illustrate this for (a), leaving the other cases to the reader.

Let $y \in L(E_1) \cap R$. Since each expression $\overline{[...]}$ is either 1 or 2, it is obvious that $2n+1 \leq s_k \leq 3n+1$ for all k. Consider the X_1-assignment obtained from y via the p_{1i} as just described. Note that $s_k = 3n+1$ iff "2" contributes to each of the n terms of the sum. Suppose that $x_{1i} \in C_k$. Then $\overline{[x_{1i} \in C_k]}$ has (integer) value 1. Therefore, the ith term contributes "2" to the sum iff $p_{1i} = 1$ iff x_{1i} is *true*. Similarly, if $\neg x_{1i} \in C_k$, then the ith term contributes "2" to the sum iff $p_{1i} = 0$ iff x_{1i} is *false*. It follows that $s_k = 3n+1$ iff C_k is not killed.

Remember that our goal was to show

$$\forall X_1 \exists X_2 (G(X_1, X_2) = 0)$$
$$\text{iff}$$
$$L(E_1) \cap R \subseteq L(E_2) \cap R.$$

Since $G(X_1, X_2) = 0$ iff all clauses are killed, it is easy to prove "only if" from (a) and (d), while "if" follows from (b) and (c). As noted above, this proves (1). Finally, from (1) we have

$$\exists X_1 \forall X_2 (G(X_1, X_2) = 1)$$
$$\text{iff}$$
$$L(E_1 \cup E_2 \cup \overline{R}) \neq L(E_2 \cup \overline{R}).$$

□

7 Inequivalence for Expressions with Star

Let EXPSPACE denote the class of decision problems solvable by deterministic Turing machines within space d^n for some constant d. The problem NEC-$\{\cup, \cdot, *, \cap\}$ is known to be EXPSPACE-complete. This was first proved by Hunt [Hunt73] who also proved that this problem requires space c^{2^n} for some constant $c > 1$. The proof was simplified by Fürer [Furer30] and the lower bound was improved to c^n. We show in this section that EXPSPACE-completeness of NEC and INEQ holds also if the intersection operator is replaced by the interleaving operator.

Theorem 7.1 *INEQ-$\{\cdot, \cup, *, \|\}$ and NEC-$\{\cdot, \cup, *, \|\}$ are EXPSPACE-complete.*

PROOF:

(1) INEQ-$\{\cup,\cdot,*,|\} \in$ EXPSPACE.

Given $\{\cup,\cdot,*,|\}$-expressions E_1 and E_2 of length at most n, it is easy to build NFA's M_1 and M_2 with $O(2^n)$ states which accept $L(E_1)$ and $L(E_2)$, respectively. The product construction of Section 2 is used for $|$. Using the simulation method described in Section 4, it is easy to show that equivalence of NFA's can be decided by a nondeterministic Turing machine within space proportional to the size of the NFA's (Thm. 13.14 of [HU79] uses a similar proof).

(2) NEC-$\{\cup,\cdot,*,|\}$ is EXPSPACE-hard.

Fürer proves in [Furer80] the EXPSPACE-hardness of NEC-$\{\cup,\cdot,*,\cap\}$ by doing a generic reduction from an exponential-space Turing machine. This proof will serve as a basis for our proof. We will show that by adding new format requirements for words describing accepting computations we can simulate the intersection operator by the interleaving operator.

The key in Fürer's proof is that the there is a succinct (i.e., its length is $O(n)$) expression with intersection r_n which describes the language $P_n = \{w\gamma w^{\mathcal{R}} : w \in \Gamma^*, |w| = n, \gamma \in \Gamma\}$, where Γ is a finite alphabet and where $w^{\mathcal{R}}$ denotes the reverse of w. r_n can be defined inductively as follows:

$$r_0 = \Gamma$$

$$r_{i+1} = \Gamma \cdot r_i \cdot \Gamma \cap \bigcup_{\gamma\in\Gamma} \gamma \cdot \Gamma^* \cdot \gamma.$$

Thus r_n contains n nested occurrences of "$\cap$". We now show that we can describe a language similar to P_n by an expression which contains n nested occurrences of "$|$", provided that words are required to have a certain restricted format. Let $\Gamma = \{\gamma_1, \ldots, \gamma_l\}$. Let c be a symbol not in Γ. If $w = w_1 w_2 \ldots w_n$ where $w_i \in \Gamma$ for $1 \le i \le n$, and if k is a positive integer, then

$$w^{(k)} = w_1^k c^k w_2^k c^k \ldots w_n^k c^k.$$

Also, $\epsilon^{(k)} = \epsilon$. Letting A be any language over Γ, define $A^{(k)} = \{ w^{(k)} : w \in A \}$.

Words having the required format are in the set $R^{(k)}$ defined as:

$$R^{(k)} = (\Gamma^*)^{(k)} = ((\gamma_1^k \cup \ldots \cup \gamma_l^k)c^k)^*.$$

Let the expression s_n be defined inductively as follows:

$$s_0 = \Gamma \cdot c$$

$$s_{j+1} = (\Gamma^{2^{j+1}} \cdot c^{2^{j+1}} \cdot s_j \cdot \Gamma^{2^{j+1}} \cdot c^{2^{j+1}}) \mid (\bigcup_{\gamma\in\Gamma} \gamma \cdot c \cdot (\Gamma \cdot c)^* \cdot \gamma \cdot c).$$

Note that the length of s_j is $O(j^2)$.

We now claim that those words in s_j which are restricted to be in $R^{(2^j+1)}$ describe a language similar to the one described by r_j. This will be proved in Lemma 7.4, following two preliminary lemmas. The first lemma follows immediately from the definition of the s_j.

Lemma 7.2 *If $w \in L(s_j)$, then $w = \gamma y c$ for some $\gamma \in \Gamma$ and $y \in (\Gamma \cup \{c\})^*$.*

To state the second lemma, we need a definition. If $w \in (\Gamma \cup \{c\})^*$, let $M(w)$ be the maximum length of a subword u of w such that $u \in \Gamma^* \cup \{c\}^*$. Note that if $w \in L(y \mid z)$, then $M(w) \leq M(y) + M(z)$.

Lemma 7.3 *If $w \in L(s_j)$, then $M(w) \leq j + 1$.*

PROOF: The proof is by induction on j. The basis $j = 0$ is obvious. Assume the lemma is true for some j. To prove the induction step, let $w \in L(s_{j+1})$. Then $w \in L(y \mid z)$ for some

$$y \in L(\Gamma^{j+1} \cdot c^{j+1} \cdot s_j \cdot \Gamma^{j+1} \cdot c^{j+1}) \text{ and } z \in L(\bigcup_{\gamma \in \Gamma} \gamma \cdot c \cdot (\Gamma \cdot c)^* \cdot \gamma \cdot c).$$

By Lemma 7.2 and the induction hypothesis, $M(y) \leq j + 1$. It is obvious that $M(z) = 1$. So $M(w) \leq j + 2$. □

We can now prove the connection between $L(s_j)$ and P_j.

Lemma 7.4 $L(s_j) \cap R^{(j+1)} = (P_j)^{(j+1)}$.

PROOF: Our proof will be by induction on j.

Induction Basis: If $j = 0$, then $L(s_0) \cap R^{(1)} = \Gamma \cdot c = (P_0)^{(1)}$.

Induction Hypothesis: $L(s_j) \cap R^{(j+1)} = (P_j)^{(j+1)}$.

Induction Step: We want to show that $L(s_{j+1}) \cap R^{(j+2)} = (P_{j+1})^{(j+2)}$. It is easy to see that $(P_{j+1})^{(j+2)} \subseteq L(s_{j+1}) \cap R^{(j+2)}$, so we only show the opposite inclusion. Any word in $R^{(j+2)}$ is made of "chunks" consisting of $j + 2$ identical symbols of Γ followed by $j + 2$ c's. Let $xyz \in L(s_{j+1}) \cap R^{(j+2)}$, where x is the first chunk, z is the last chunk, and y is all chunks in between. Using Lemma 7.2, the first and the last chunk, x and z, must result from the interleaving of $\Gamma^{j+1} \cdot c^{j+1}$ and $\gamma \cdot c$. Moreover, since the same γ must be used for both x and z, we have $x = z$. The word y must result from the interleaving of some $y' \in L(s_j)$ and $(\Gamma \cdot c)^*$. From this and Lemma 7.3, we can conclude that only those words $y' \in L(s_j)$ which are also in $R^{(j+1)}$ can be used to form a word in $R^{(j+2)}$. By the induction hypothesis, $y' \in (P_j)^{(j+1)}$. Since y is a concatenation of chunks, it follows that $y \in (P_j)^{(j+2)}$. Since $x = z$, it follows that $xyz \in (P_{j+1})^{(j+2)}$. □

We now describe the reduction. For simplicity, we do the reduction from a deterministic one-tape Turing machine M with space bound $2^n - 3$. The extension to general exponential space bounds is straightforward. Let M have tape alphabet T, state set S, accepting states F, and start state q_0. Let x be an input to M, and let $n = |x|$ and $m = 2^n - 1$. Let $\Sigma_{ID} = T \cup (S \times T) \cup \{\$\}$. An ID of M is a word of length m in $(\Sigma_{ID})^*$ of the form $\$u(q, \alpha)v\$$ where $uv \in T^*$; this ID means that the string $u\alpha v$ is written on the tape, and M is in state q scanning the symbol α. (This representation of ID's is slightly different than the one used in [Furer80], but it is convenient for our purposes.)

As in [Furer80], we use "marked binary numbers" to index the symbols of an ID. A *marked binary number* is a word over the alphabet $\{0, \underline{0}, 1, \underline{1}\}$ in the language described by the expression $(0 \cup 1)^* \underline{1}\underline{0}^* \cup \underline{0}^*$; i.e., the rightmost (lowest order) 1 is marked, as well as all 0's to the right of this 1; and in the representation of 0, all 0's are marked. For integer j

with $0 \leq j \leq m$, let $[j]$ denote the length-n marked binary representation of j. The marking allows the successor relation to be tested locally as follows. Define $\mathrm{succ}(0) = \mathrm{succ}(\underline{0}) = \{0, \underline{1}\}$ and $\mathrm{succ}(1) = \mathrm{succ}(\underline{1}) = \{1, \underline{0}\}$. If $y_n \ldots y_1 = [j]$ and $z = z_n \ldots z_1$ is a marked binary number of length n, then $z = [j+1 \bmod 2^n]$ iff $z_j \in \mathrm{succ}(y_j)$ for $1 \leq j \leq n$.

Let $\Sigma = \Sigma_{\mathrm{ID}} \cup \{0, \underline{0}, 1, \underline{1}, \#, \&, c\}$ and $\Gamma = \Sigma - \{c\}$. The accepting computation of M on input x, provided that it exists, is represented by the following word $a \in \Sigma^*$:

$$a = (a')^{(n+1)}$$

where

$$\begin{aligned} a' \;=\; & \&\,[0]^R \,\#\, [0] \,\&\, [1]^R\, a_{1,1}\, [1] \,\&\, [2]^R\, a_{1,2}\, [2] \,\&\, \ldots \\ & \ldots\, [m]^R\, a_{1,m}\, [m] \,\&\, [0]^R \,\#\, [0] \,\&\, [1]^R\, a_{2,1}\, [1] \,\&\, \ldots \\ & \ldots\, [m]^R\, a_{2,m}\, [m] \,\&\, [0]^R \,\#\, [0] \,\&\, [1]^R\, a_{3,1}\, [1] \,\&\, \ldots \\ & \qquad\qquad \ldots\ \ldots \\ & \ldots\, [m]^R\, a_{k,m}\, [m] \,\&\, [0]^R \,\#\, [0] \,\& \end{aligned} \tag{2}$$

where $a_i = a_{i,1} a_{i,2} \ldots a_{i,m}$ is the ith ID in the computation of M on input x. (In [Furer80], the word a' is used to represent an accepting computation.) We say that a word $a \in \Sigma^*$ has the *correct framework* if $a = (a')^{(n+1)}$ for some word a' as in (2), where $a_{i,1} = a_{i,m} = \$$ for $1 \leq i \leq k$, but where the symbols $a_{i,j}$, for $1 \leq i \leq k$ and $1 < j < m$, can be any symbols of Σ_{ID}.

We now simply have to enumerate the mistakes which imply that a word is not a computation of M on input x. Each type of mistake is described by an expression. Letting E_x be the union of these expressions, it follows that $L(E_x) \neq \Sigma^*$ iff M accepts x. The length of E_x will be $O(n^2)$. The following enumeration of mistakes was chosen to highlight the more interesting and original parts of the construction. For example, we consider "not having the correct framework" to be a single type of mistake, even though this could be broken down into several types of lower level mistakes.

0. The expression E_0 describes all words not in $R^{(n+1)}$.

1. When restricted to words in $R^{(n+1)}$, the expression E_1 describes all words which do not have the correct framework.

2. When restricted to words having the correct framework, E_2 describes all words such that a_1 is not the initial ID of M on input x (i.e., $a_1 \neq \$(q_0, x_1) x_2 \ldots x_n B^{m-n-2} \$$ where B denotes the blank tape symbol), or such that no symbol of the form (q, α) appears where q is an accepting state.

3. When restricted to words having the correct framework, E_3 describes all words which have a "computation error", i.e., words such that some $a_{i+1,j}$ with $1 < j < m$ does not follow correctly from $a_{i,j-1}, a_{i,j}, a_{i,j+1}$ by the transition rules of M.

We first describe E_3 in detail, since it is the more interesting part of the construction. Let $f : (\Sigma_{\mathrm{ID}})^3 \to \Sigma_{\mathrm{ID}}$ be such that, in any correct computation, $a_{i+1,j} = f(a_{i,j-1}, a_{i,j}, a_{i,j+1})$ for all $1 \leq i < k$ and $1 < j < m$. All such occurrences can be found, because $a_{i,j}^{(n+1)}$ is to

the left of $([j])^{(n+1)}$, $a_{i+1,j}^{(n+1)}$ is to the right of $([j]^R)^{(n+1)}$, and there is exactly one block of #'s between them. Thus the relevant part of a word representing an accepting computation must look as the following:

$$\begin{gathered}
\ldots \\
([j-1]^R)^{(n+1)} \cdot (a_{i,j-1})^{(n+1)} \cdot ([j-1])^{(n+1)} \cdot \&^{(n+1)} \cdot \\
([j]^R)^{(n+1)} \cdot (a_{i,j})^{(n+1)} \cdot ([j])^{(n+1)} \cdot \&^{(n+1)} \cdot \\
([j+1]^R)^{(n+1)} \cdot (a_{i,j+1})^{(n+1)} \cdot ([j+1])^{(n+1)} \cdot \&^{(n+1)} \\
\ldots \\
\#^{(n+1)} \\
\ldots \\
([j]^R)^{(n+1)} \cdot (a_{i+1,j})^{(n+1)} \cdot ([j])^{(n+1)} \\
\ldots
\end{gathered}$$

Now we can construct an expression similar to the one in Lemma 7.4 to denote all wrong computation steps. Let μ, ν, ξ be symbols in Σ_{ID}, corresponding to $a_{i,j-1}$, $a_{i,j}$, $a_{i,j+1}$, respectively.

$$t_0(\xi) = (\Gamma \cdot c)^{n+1} \cdot \xi \cdot c \cdot ((\Gamma - \{\#\}) \cdot c)^* \cdot \# \cdot c \cdot ((\Gamma - \{\#\}) \cdot c)^*$$

$$t_{j+1}(\xi) = (\Gamma^{j+1} \cdot c^{j+1} \cdot t_j(\xi) \cdot \Gamma^{j+1} \cdot c^{j+1}) \mid (\bigcup_{\gamma \in \Gamma} \gamma \cdot c \cdot (\Gamma \cdot c)^* \cdot \gamma \cdot c).$$

As in the proof of Lemmas 7.2–7.4, the following can be proved by induction on j.

Lemma 7.5 *$w \in L(t_j(\xi)) \cap R^{(j+1)}$ iff there exist words $z \in \Gamma^j$, $u \in \Gamma^{n+1}$, and $v, y \in (\Gamma - \{\#\})^*$, such that $w = (z\,u\,\xi\,v\,\#\,y\,z^R)^{(j+1)}$.*

Now E_3 is the union, over all $\mu, \nu, \xi \in \Sigma_{\mathrm{ID}}$, of:

$$\Sigma^* \cdot \mu^{(n+1)} \cdot \Sigma^{4n^2+6n+2} \cdot \nu^{(n+1)} \cdot t_n(\xi) \cdot (\Sigma_{\mathrm{ID}} - \{ f(\mu, \nu, \xi) \})^{(n+1)} \cdot \Sigma^*.$$

As mentioned, E_0 denotes the mistake of a word not being in $L(R^{(n+1)})$. We split E_0 in four categories, i.e., $E_0 = \bigcup_{j=1}^{4} E_{0j}$. E_{01} (E_{02}) takes care of the case of a block being too short (long), E_{03} describes the case where not every other block is composed of c's, and E_{04} describes words which start and end wrong:

$$E_{01} = \bigcup_{\sigma \in \Sigma} (\epsilon \cup (\Sigma^* \cdot (\Sigma - \{\sigma\}))) \cdot \sigma \cdot (\sigma \cup \epsilon)^{n-1} \cdot (\epsilon \cup ((\Sigma - \{\sigma\}) \cdot \Sigma^*))$$

$$E_{02} = \bigcup_{\sigma \in \Sigma} \Sigma^* \cdot \sigma^{n+2} \cdot \Sigma^*$$

$$E_{03} = (\epsilon \cup (\Sigma^* \cdot c)) \cdot (\Sigma - \{c\}) \cdot \Sigma^n \cdot (\Sigma - \{c\}) \cdot \Sigma^*$$

$$E_{04} = c \cdot \Sigma^* \cup \Sigma^* \cdot (\Sigma - \{c\}).$$

The expression E_1 which describes all framework mistakes is conceptually not difficult, since the checking can all be done "locally", i.e., the symbols to be checked are within distance $O(n^2)$. This expression can be based on the ones given in [Furer80]. For these reasons, we do not write E_1 in detail. For illustration, we write an expression for one type

of framework error where the marked binary numbers embedded in the computation a are not incremented correctly. The relevant part of a word having the correct framework is

$$\ldots ([j])^{(n+1)} \&^{(n+1)} ([j+1 \bmod 2^n]^R)^{(n+1)} \ldots .$$

Let $D = \{0, \underline{0}, 1, \underline{1}\}$, and let $r^+ = r \cdot r^*$. Recalling that we can restrict attention to words in $R^{(n+1)}$, the following describes all "incrementing mistakes":

$$\bigcup_{j=0}^{n-1} \bigcup_{\sigma \in D} \Sigma^* \cdot \sigma^+ \cdot c^+ \cdot (D^+ \cdot c^+)^j \cdot \&^+ \cdot c^+ \cdot (D^+ \cdot c^+)^j \cdot (D - \mathrm{succ}(\sigma))^+ \cdot c^+ \cdot \Sigma^*.$$

The interested reader can easily complete the construction of E_1 by writing expressions of length $O(n^2)$ for the other types of framework errors.

The construction of E_2 is also straightforward and is left to the reader. □

Since the length of E_x is $O(n^2)$, a lower bound on space complexity follows by a standard argument (e.g., pg. 418 in [AHU74]).

Corollary 7.6 *There is a constant $c > 1$ such that no deterministic Turing machine with space bound $c^{\sqrt{n}}$ can accept NEC-$\{\cup, \cdot, *, |\}$ or INEQ-$\{\cup, \cdot, *, |\}$.*

Note that this lower bound ($c^{\sqrt{n}}$) does not match the upper bound (d^n).

By using a coding like the one described in the proof of Lemma 5.4, it can be shown that Theorem 7.1 and Corollary 7.6 remain true for expressions over an alphabet of size 3.

Acknowledgements: The first author wishes to thank his advisor Paris Kanellakis for the help during this work and the "Alice Mayer Foundation for Gifted Swiss Students of Jewish Hungarian Descent" for partial financial support.

References

[AHU74] A.V. AHO, J.E. HOPCROFT, AND J.D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[Eilen74] S. EILENBERG, *Automata, Languages, and Machines*, Vol. A, Academic Press, New York, 1974.

[Furer80] M. FÜRER, The complexity of the inequivalence problem for regular expressions with intersection, *Proc. 7th ICALP, Lecture Notes in Computer Science*, Vol. 85, Springer-Verlag, New York, 1980, pp. 234–245.

[HU79] J.E. HOPCROFT AND J.D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[Hunt73] H.B. HUNT III, The equivalence problem for regular expressions with intersection is not polynomial in tape, Tech. Rep. TR 73-161, Cornell University, 1973.

[HRS76] H.B. HUNT III, D.J. ROSENKRANTZ, AND T.G. SZYMANSKI, On the equivalence, containment, and covering problems for the regular and context-free languages, *J. Comput. System Sci.* **12** (1976), 222–268.

[IPC85] O.H. IBARRA, M.A. PALIS, AND J.H. CHANG, On efficient recognition of transductions and relations, *Theoretical Computer Science* **39** (1985), 89–106.

[KS90] P.C. KANELLAKIS AND S.A. SMOLKA, CCS expressions, finite state processes, and three problems of equivalence, *Information and Computation* **86** (1990), 43–68.

[Miln80] R. MILNER, *A Calculus of Communicating Systems, Lecture Notes in Computer Science*, Vol. 92, Springer-Verlag, New York, 1980.

[Miln84] R. MILNER, A complete inference system for a class of regular behaviors, *J. Comput. System Sci.* **28** (1984), 439–466.

[Stock74] L.J. STOCKMEYER, The complexity of decision problems in automata theory and logic, Tech. Rep. TR-133, MIT, Project MAC, 1974.

[Stock77] L.J. STOCKMEYER, The polynomial-time hierarchy, *Theoretical Computer Science* **3** (1977), 1–22.

[StM73] L.J. STOCKMEYER AND A.R MEYER, Word problems requiring exponential time, *Proc. 5th ACM Symp. on Theory of Computing* (1973), 1–9.

[vLN82] J. VAN LEEUWEN AND M. NIVAT, Efficient recognition of rational relations, *Information Processing Letters* **14** (1982), 34–38.

[Wrath77] C. WRATHALL, Complete sets and the polynomial-time hierarchy, *Theoretical Computer Science* **3** (1977), 23–33.